Cheat Detection for MMORPG on P2P Environments

Takato Izaiku, Shinya Yamamoto, Yoshihiro Murata,
Naoki Shibata[†], Keiichi Yasumoto, Minoru Ito

Graduate School of Information Science,
Nara Institute of Science and Technology
Ikoma, Nara 630-0192, Japan
{takato-i,shiny-ya,yosihi-m,yasumoto,ito}@is.naist.jp

[†] Department of Information Processing and
Management, Shiga University
Hikone, Shiga 522-8522, Japan
shibata@biwako.shiga-u.ac.jp

ABSTRACT

In recent years, MMORPG has become popular. In order to improve scalability of game system, several P2P-based architectures have been proposed. However, in P2P-based gaming architecture, cheats by malicious players may more likely occur than traditional centralized architecture, since most of game data is handled by player nodes. In this paper, we propose a new method for detecting cheat in MMORPG which supposes typical P2P-based event delivery architecture where the entire game space is divided into subareas and a responsible node (selected from player nodes) delivers each event happened in the subarea to player nodes there every predetermined time interval called timeslot. In the proposed method, we introduce multiple monitor nodes (selected from player nodes) which monitor the game state and detect cheat when it happens. In order to allow monitor nodes to track the correct game state for the corresponding subarea, we let monitor nodes and a responsible node retain a random number seed and player nodes send their events not only to responsible node but also monitor nodes so that the monitor nodes and the responsible node can uniquely calculate the latest game state from the previous game state and game events which happened during the current timeslot. Either responsible node, monitor nodes or player nodes can detect cheat by comparing hash values of game state which are retained by those nodes periodically, and role back events happened since the last correct game state. Through experiments in PlanetLab, we show that our method achieves practical performance to detect cheats.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.4 [**Computer-Communication Networks**]: Distributed System

General Terms

Design, Security, P2P, MMORPG

Keywords

Distributed System, Cheating, Massively Multiplayer Online Gaming, Peer to Peer

1. INTRODUCTION

Thanks to recent progress of network technology, we are now able to enjoy network gaming at home using PC, video game console, and so on. However, due to popularization of network game, servers for Massively Multiplayer Online Role Playing Game (MMORPG) have been highly overloaded. Accordingly, some load distribution methods based on P2P technology have been proposed. While these methods have good scalability to the number of players, they suffer from preventing cheat by malicious players since most of game data such as game state and events are managed by player nodes. Thus, a new method for preventing cheat on P2P-based gaming architecture is desired.

There are several research efforts for preventing cheat on P2P-based game. In [6], Lock-Step protocol is proposed for preventing Look Ahead Cheat in which a responsible node (which is a special node delivering game events happened in a subarea to player nodes there, although it is also a player node) decides its event after seeing events taken by other players. In Lock-Step protocol, in order to prevent Look Ahead Cheat, each player node (including the responsible node) is required to send the hash value of an event which the corresponding player wants to issue to the responsible node in advance, and then the responsible node gathers the events of all players. However, Lock-Step protocol cannot prevent packet falsifying by malicious responsible node. Also, it cannot detect cheats such that malicious responsible node intentionally disposes player's events. Furthermore, with this protocol, delay of the entire game becomes large depending on the largest delay among player nodes, due to synchronization when gathering all events from player nodes.

In [2], NEO (New Event Ordering) protocol is proposed. This protocol takes a vote among all players based on Lock-Step protocol. While this protocol does not need responsible node, since all player nodes share the game state by broadcasting event to each other, this protocol may exclude some player nodes with large delay or low available bandwidth from voting, in order to improve efficiency. Also, this protocol requires each node to broadcast event to all the other nodes, so required bandwidth increases. Moreover, since this protocol does not determine the order of received

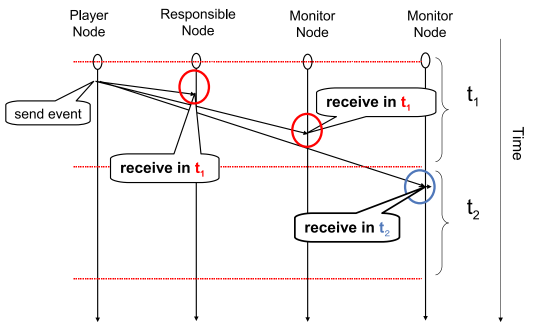

Figure 1: Allocation of event list to timeslot

events, game state can become inconsistent among player nodes.

[7] proposes a method to detect cheat by collating game logs. When a player logs out from the game system, this method compares the logs for transmitted packets by responsible node and received packets by player node. However, since this method can detect cheat only when players log out, it takes long time to detect cheat and is difficult to correct game state while the malicious player is committing cheats in the game system.

In [5], various cheats are classified from three points of view: weakness of the system; types of cheat performer; and means of cheat. Also, vulnerability against cheat is classified into four categories: weakness of networks (e.g. DoS attack or modification of a packet); weakness of the game program (bug or modification of the program); weakness of the security (obtaining passwords); and weakness of the system (modification of OS or drivers). Among these cheats, falsification of packet is especially important in P2P-based game. Therefore, in this paper, we focus on this type of cheat.

In this paper, we propose a new method for detecting cheat in MMORPG which supposes typical P2P-based event delivery architecture such as [4, 8] where the entire game space is divided into subareas and a *responsible node* (selected from player nodes) delivers each event happened in the subarea to player nodes there every predetermined time interval called *timeslot*. In the proposed method, we introduce multiple monitor nodes (selected from player nodes) which monitor the game state and detect cheat when it happens. In order to allow monitor nodes to track the correct game state for the corresponding subarea, we let monitor nodes and a responsible node retain a random number seed and player nodes send their events not only to responsible node but also to monitor nodes so that the monitor nodes and the responsible node can uniquely calculate the latest game state from the previous game state and game events which happened during the current timeslot. Either responsible node, monitor nodes or player nodes can detect cheat by comparing hash values of game state which are retained by those nodes periodically, and role back events happened since the last correct game state. Depending on the network topology among the responsible/monitor nodes and player nodes, the responsible/monitor nodes may receive events at different timeslots as shown in Fig. 1. In the proposed method, responsible/monitor nodes follows majority decision of which events are assigned in the timeslot.

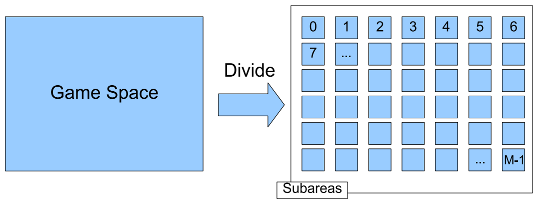

Figure 2: Division of Game Space into Subareas

By allocating multiple monitor nodes for each subarea, collusion between the responsible node and monitor nodes can be detected by majority voting. Since only part of nodes participates in voting, the amount of messages can be kept much lower than NEO protocol. Moreover, in the proposed method, since comparison of game state is performed in sufficiently long period, the problem in [7] can be avoided.

In order to show usefulness of our method, we have implemented a prototype and conducted some experiments in a WAN testbed called PlanetLab[9]. In the environment where traffic delay between player node and responsible node is about 200 msec, our method could detect cheat in 1 second, and correct game state in 8 seconds, while keeping the extra traffic at player nodes low enough.

2. PROPOSED METHOD

In this section, we first describe the target P2P-based game system in which we want to detect cheats, and then we describe the proposed method.

2.1 Target Game System

The proposed method is constructed on top of typical P2P-based game system such as [4, 8]. In such a game system, player nodes first contact with a *lobby server* when logging in. Lobby server is used only for management of player accounts, which is just a light job and can be distributed to multiple computers. Other heavier jobs like event processing and delivering are performed by player nodes, and this makes it possible to realize MMORPG without dedicated servers or high speed network. In P2P-based game system, the game space is usually divided into subareas as shown in Fig. 2, and a *responsible node* is selected from player nodes and assigned to each subarea, so that the global game state is maintained separately as the set of sub states for those subareas.

According to the game system in [8], we assume that time is divided into discrete units. We call this unit *timeslot*. Timeslot t_n is a range of time $[T_0 + \Delta * n, T_0 + \Delta * (n+1))$, where T_0 is the starting time of game, Δ is a constant value, and n is an integer number. The game space contains multiple *objects*. Moving objects including player characters and other static objects are all objects. An object has attributes like position in game space, state and so on.

For a subarea v and timeslot t_n, a tuple of attributes of all objects in v is called the game state on v at t_n, and denoted as $GS(t_n, v)$. An *event* is an action taken by a player or an incident which affects the game state of the next timeslot. An event generated by player p_i at timeslot t_n is denoted as $E(t_n, p_i)$. In this paper, we assume that each player generates at most one event in a timeslot. $EL(t_n, v)$ is the set of all events generated in a subarea v during timeslot t_n, which is called *event list*.

2.2 Target Cheats to be Detected

In the above P2P-based game system, the responsible node plays an important role to deliver events to player nodes. So, if a malicious player node is selected as the responsible node, it can easily falsify the event list before sending it so that some player nodes will be disadvantaged or the responsible node will be advantaged.

Even when the responsible node is fair, it has no means to prevent malicious player nodes from sending impossible events.

The proposed method provides a way to detect the above two types of cheats as well as criminals and to correct the falsified game state to the valid one by rollback.

2.3 Assumption

We assume that all nodes participating the game has a pair of secret and public keys, and nodes are able to obtain a public key of any other node. We also assume that all player nodes have a random number seed after they log into the game system.

We regard that the whole game system is constructed as a set of finite state machines (FSMs) where each FSM corresponds to the sub game system on a subarea and transits every timeslot. That is, each state of the FSM for subarea v is $GS(t_n, v)$, and $GS(t_{n+1}, v)$ can be uniquely determined by $GS(t_n, v)$ and $EL(t_n, v)$. Responsible node receives events from player nodes on its responsible subarea, and generates an event list every timeslot. Then, the responsible node transmits the event list to all player nodes in the subarea. Each player node receives the event list from the responsible node, and it calculates the next state. Player nodes send events for the next time slot to the responsible node, and the game proceeds by repeating these steps.

2.4 Method for cheat detection

In order to detect cheats, the proposed method uses multiple *monitor nodes* for each subarea which are selected from player nodes similarly to the responsible node, and lets each player node sends its event not only to the responsible node but also to the monitor nodes as shown in Fig. 3. The responsible node and the monitor nodes compare the game states retained by them and events received by all players periodically.

First of all, we give an explanation when a player node tries to forge an impossible event which is not allowed at the current game state. A player trying to use an item which is not possessed by the player is an example of this case. This can be prevented at a responsible/monitor nodes by defining the next game state of FSM transit by an event list with impossible event to be identical to the state transit by the event list excluding the event.

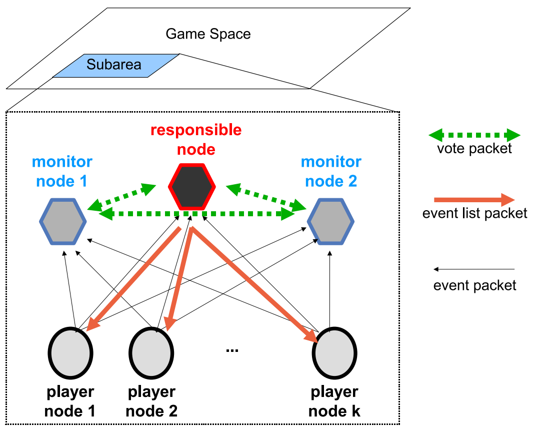

Figure 3: Communication between player nodes and responsible/monitor nodes

Even if neither nodes commit cheating, an event sent by a player node may be received by the responsible node and the monitor nodes at different timeslots due to network delay. If some events are assigned in different timeslots among responsible node and monitor nodes, they cannot track the same state in their FSMs and thereby cannot detect cheat.

In the proposed method, basically the responsible node and the monitor nodes follow majority decision on which timeslot each event belongs to. However, conducting majority voting among nodes every timeslot may not be practical in terms of delay and message overhead. So, we reduce the number of voting by the following way. When a player node sends an event, it attaches to the message the timeslot at which the event is sent. If the event is not received in the same timeslot and the delay is within a predetermined range (called *discretion range*), the responsible node determines which timeslot the event corresponds to. If the delay exceeds discretion range, voting takes place. The discretion range for each player node is adapted according to the average delay.

In the above mechanism, some malicious player nodes may try to disturb the smooth progress of game by intentionally delaying events to be sent so that voting frequently takes place. This can be avoided by prolonging the discretion range.

The responsible node may falsify the event list. This can be classified into two cases: the case that the responsible node falsifies the event list, and the case that the responsible node intentionally ignores some events received from some player nodes. In order to detect the former case, the proposed method uses digital signature. When player node sends event to the responsible node, it adds digital signature to the event message. In the proposed method, the responsible node and the monitor nodes compare events received by player nodes periodically, and authenticity for these events can be confirmed by checking the digital signatures. If the responsible node and the monitor nodes have authentic digital signatures for all events, in spite of having different game states, some player node is considered

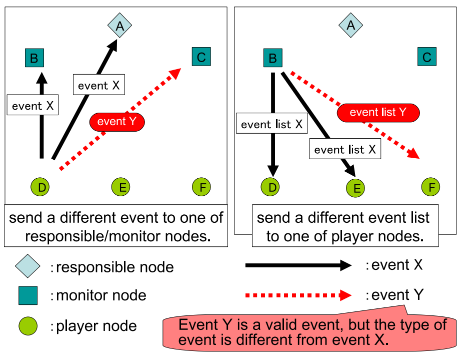
A
B
C
D
E
F
event X
event X
event Y
event list X
event list X
event list Y
send a different event to one of responsible/monitor nodes.
send a different event list to one of player nodes.
:responsible node
:monitor node
:player node
:event X
:event Y
Event Y is a valid event, but the type of event is different from event X.

Figure 4: Possible cheat to attack proposed method

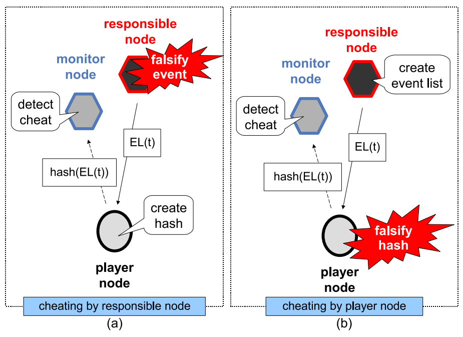

Figure 5: :Detection of cheat by responsible node and by player node

to send different events to them at the same timeslot. An example of this situation is depicted in Fig. 4. Otherwise, the responsible node or one of monitor nodes is considered to commit a cheat. Our method detects this case by letting each player node attach the hash value of the event list received at the last timeslot to an event which the player wants to generate at the current timeslot. An example of this situation is shown in Fig. 5 (a). The latter case can be detected by majority voting among the responsible node and the monitor nodes.

Player node may try to entrap the responsible node or one of the monitor nodes, by sending a different event to the node as shown in Fig. 5 (b). Our method can detect this case by letting responsible and monitor nodes to compare events with signatures received from each player node.

An outsider node may intrude between two nodes, and impersonate one of the nodes. This can be prevented by the lobby server issuing a public key certificate for each node. In order to prevent impersonating the lobby server, an existing method to prevent impersonating utilized on web is used.

In order to add uncertainty to the game, a mechanism to generate random numbers is required. In order to maintain consistency among player nodes, the responsible node, and the monitor nodes, all nodes have to retain a common random number seed. Since all player nodes know the seed, it may be possible to predict random numbers of the future, and thus it can, for example, attack at advantageous timing. This can be avoided by making each node to send a random number taken from its environment (e.g., `/dev/urandom` in Linux) as a part of event, and all nodes update the seed based on the numbers.

2.5 Other issues

User Moving between Subareas

When a player character moves into another subarea, game state regarding to the player has to be handed over to the new subarea. In this case, messages are exchanged between the two responsible nodes of the neighboring subareas. Monitor nodes of those subareas also exchange messages. And then, the game states are compared between the responsible nodes and monitor nodes.

A player may try to disturb the smooth progress of game by frequently crossing the border between two subareas so that voting frequently takes place. This can be avoided by making the responsible nodes tentatively progress the game assuming that there is no cheat, and perform voting afterward. If cheating is detected at the vote, rollback to the last correct game state is carried out.

Event Spreads over Multiple Subareas

Events spread over multiple subareas are treated similarly to the case of player moving between subareas. In this case, the responsible node at the originating subarea sends event to the responsible nodes at the influenced subareas. Monitor nodes of those subareas also exchange messages. Cheating is detected by comparing the game states between the responsible nodes and monitor nodes.

Node Failures

P2P based game system has to be aware of abrupt node failures. If the responsible node fails, the role of responsible node must be taken over by one of the monitor nodes. As for the failure of a monitor node, extra monitor node is allocated.

Preventing Collusion

If more than two nodes of the responsible node or the monitor nodes make collusion (e.g., under control of a single person or group), more than two nodes may participate on cheating. In this case, voting may not work as expected. This can be avoided by choosing unrelated nodes as monitor nodes and responsible node, for example, based on IP address.

3. ANALYSIS OF TRAFFIC OVERHEAD

In the proposed method, each player node is required to send event messages not only to a responsible node but also to the associated monitor nodes. In order to investigate whether or not our proposed method is practical, we analyze traffic amount which typical MMORPG will produce with the proposed method.

3.1 Supposed Environment

We assume that each event issued by a player is sent as a packet. We call it *event packet.* Event packet consists of header part and payload part. The header part should contain sender (player) ID, packet (timeslot) ID, event type and location of event. So, we can estimate that the size of header part would be about 16 bytes if each field uses 4 bytes. The payload part contains the event value, complementary information of event, or so on, and its size varies depending on games. In general, the size of the payload part could be estimated around 16 to 48 bytes.

According to [3], the perceptually adequate response time (i.e., the time from event occurrence till getting the updated game state) in the network game is less than 100 to 300 msec. If the response time is larger than this threshold, game player feels strange or uncomfortable.

According to the above discussion, we suppose that the interval of timeslot is 200 msec, size of event packet is 64bytes and the maximum number of player nodes in each subarea is 100.

In our event delivery architecture [8] without monitor nodes, each player transmits to a responsible node an event packet every timeslot (requiring 64bytes/200msec = 2.56Kbps uplink bandwidth), and a responsible node receives event packets from up to 100 player nodes every timeslot and delivers the aggregated event list to all of the player nodes (requiring 64byte × 100/200msec = 256Kbps downlink bandwidth and 64byte × 100 × 100/200msec = 25.6Mbps uplink bandwidth, for the maximum).

3.2 Extra Traffic at Player Node

In the above environment, if the proposed cheat detection mechanism is utilized with two monitor nodes for each subarea, the extra traffic amount which each player may produce per second will be 64byte × 2/200msec = 5.12Kbps. So, as long as the number of monitor nodes is small, traffic overhead by adding monitor nodes is small enough.

3.3 Traffic Dealt with Monitor Node

In the proposed method, responsible node delivers all events taken place in a subarea to player nodes, but monitor nodes need not. Therefore, extra traffic is not produced from monitor nodes to player nodes.

The traffic amount which each monitor node may receive is 64byte × 100/200ms = 32000byte/sec = 256Kbps, which is equivalent to the traffic which responsible node receives. This bandwidth can easily be achieved by ADSL, CATV or FTTH. So, most of player nodes connecting to such a broadband network can be selected as monitor node.

In the proposed method, responsible node and the associated monitor nodes periodically exchange messages with the latest game states which they retain, compare those game states, and accept a majority decision if necessary. This message exchange happens not every timeslot but longer time interval such as every 50 timeslots (i.e., 10 sec) and so on. Moreover, since each message exchanged contains only a hash value of game state, its size is likely up to 16 bytes, and thus negligible.

If cheat is detected and responsible node and monitor nodes reach the majority decision, one monitor node with the valid game state has to send to player nodes the set of event lists which must have taken place since the last valid state so

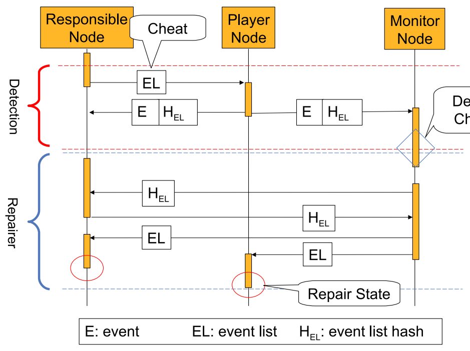

Figure 6: Message sequence to detect cheat and correct game state

that each player node can roll back the wrong events since the last valid state and transit to the latest valid state. The total message size for the rollback will be the product of the event size, the maximum number of players in a subarea and the number of timeslots in a monitoring period, that is, 64byte × 100 players × 50 timeslots = 320Kbyte. The required bandwidth is 320Kbyte/10sec = 256Kbps if we assume that cheat is detected every monitoring period. So, each monitor node may deal with up to 512Kbps traffic for reception of event packets from player nodes and transmission of packets to player nodes for rollback. We think this is practical enough for nodes connected to broadband network. The required bandwidth would actually be much smaller since cheat and rollback must happen less frequently.

4. EXPERIMENTAL RESULTS

To evaluate the proposed method, we measured time to detect a cheat and correct the game state since a cheat is committed.

4.1 The Environment of Experiment

We conducted experiments on PlanetLab [9] which is a testbed consisting of many nodes distributed over the world. The number of responsible node, monitor nodes, and player nodes for a subarea are 1, 2 and 50, respectively. In order to show that the proposed method achieves practical performance even when the delay between nodes are large, we assigned responsible node and monitor nodes on nodes in Japan, and player nodes on nodes in West Coast of the US. To simplify the experimental setting, we executed programs of five player nodes in one node of PlanetLab. Programs of responsible node and monitor nodes were executed on different nodes. Network delay between nodes within Japan was between 10 msec and 20 msec, and that between US and Japan was between 150 msec and 300 msec.

4.2 Method of Experiment

In the experiment, we let the responsible node commit cheat by sending falsified event list to player nodes every 1000 timeslots (corresponding to 300sec), and we measured time for a monitor node to detect the cheat after it is committed. In addition, we measured time to correct game state since detection of cheat. Fig. 6 shows packet flow since a cheat is committed until the state is corrected. In this experiment, we set timeslot length to be 300 msec, and let all nodes to refer to the approximately same clock by giving the difference among clocks in advance. As a result, the time difference could be regulated within 20 msec. We repeated the above experiment 20 times and each of which was continued for 20,000 timeslots (e.g., 6000sec). Each player node sends an event to the responsible node and monitor nodes every timeslot. The responsible node aggregates received events in each timeslot and sends event list to player nodes.

4.3 Results of Experiment

The average time to detect cheat and correct the game state by rollback was 1012 msec and 8023 msec, respectively. Table 1 shows the experimental result. Since the length of timeslot is 300 msec, and delay between nodes are about 200 msec, the time to detect cheating is considered to be reasonable. On the other hand, it took about 8 seconds to correct game state of player nodes. It would be practical if cheat is not so frequently committed.

5. CONCLUSION

In this paper, we proposed a method to detect cheat for P2P-based MMORPG system. The proposed method makes it possible to detect cheat of sending falsified game state by a malicious responsible node and cheat of sending impossible event by malicious player nodes, by introducing monitor nodes which periodically check if the responsible node maintains the valid game state and makes majority decision to correct the game state if necessary. Through analysis and experiments in PlanetLab, we confirmed that the proposed method can detect cheat in short time with low overhead. Part of future work will be to extend the proposed method to deal with collusion by multiple malicious player nodes.

6. REFERENCES

[1] C. Adams: "Internet X.509 Public Key Infrastructure Certificate Management Protocols," March 1999, `http://www.faqs.org/rfcs/rfc2510.html`.

[2] C. G. Dickey: "Low Latency and Cheat-proof Event Ordering for Peer-to-Peer Games," Proc. of 14th ACM Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2004), 2004.

[3] IEEE Std 1278-1993: "IEEE Standard for Distributed Interactive Simulation- Application Protocols (Revision and redesignation of IEEE Std 1278-1993)," 2002

[4] B. Knutsson, H. Lu, W. Xu and B. Hopkins: "Peer-to-Peer Support for Massively Multiplayer Games," Proc. of IEEE Infocom 2004, 2004.

[5] J. Yan: "A Systematic Classification of Cheating in Online Games," Proc. of 4th ACM Workshop on Network and System Support for Games (NetGames2005), 2005.

Table 1: Time for Cheat Detection and Rollback

number of players	detection time (msec.)	rollback time (msec.)
50	1012	8023

[6] N. E. Daughman: "Cheat-Proof Playout for Centralized and Distributed Online Games," Proc. of IEEE Infocom 2001, 2001.

[7] P. Kabus, W. Terpstra, M. Cilia, A. Buchmann: "Addressing Cheating in Distributed Massively Multiplayer Online Games," Proc. of 4th ACM Workshop on Network and System Support for Games (NetGames2005), 2005.

[8] S. Yamamoto, Y. Murata, K. Yasumoto, and M. Ito: "A Distributed Event Delivery Method with Load Balancing for MMORPG," Proc. of 4th ACM Workshop on Network and System Support for Games (NetGames2005), 2005.

[9] PlanetLab Consortium: "PlanetLab," `http://www.planet-lab.org/`.

[10] United Admins Limited: "Cheating Death," `http://www.unitedadmins.com/index.php?p=content&content=cd`.